Cryptographic Extension for Soft General-Purpose Processors with Secure Key Management

Lubos Gaspar, Viktor Fischer, Lilian Bossuet, Milos Drutarovský

2011 21st International Conference on Field Programmable Logic and Applications

Cryptographic extension for soft general-purpose processors with secure key management

Lubos GASPAR, Viktor FISCHER, Lilian BOSSUET
Université de Lyon
Laboratoire Hubert Curien, UMR 5516, CNRS
42000, Saint-Etienne, France
{*lubos.gaspar, fischer, lilian.bossuet*}*@univ-st-etienne.fr*

Milos DRUTAROVSKY
Technical University of Košice
Department of Electronics and Multimedia Communications
04120, Košice, Slovakia
Milos.Drutarovsky@tuke.sk

***Abstract*—General-purpose processors are not suitable for secure cryptographic key management. Secret keys are usually stored in the internal registers of the processor, and simple attacks on protocols, software/firmware or cache memory can often lead to key disclosure causing a system security failure. The paper presents a novel principle of processor extensions that enable secure key management. This principle is based on the creation and physical separation of three security zones: processor, cipher and key storage. In each of the three zones, the secret keys are manipulated in a different manner – as ordinary data or keys, in clear or encrypted. In order to increase security, the security zones are separated from each other on the protocol, architectural and physical level. The proposed principle is validated as extensions to both NIOS II and MicroBlaze processors. The NIOS II processor needs fewer clock cycles per data block encryption, because the security module is included in the processor's data path. The data path of the MicroBlaze is unchanged, and thus shorter, but additional clock cycles are necessary for data transfers between the processor and the security module. Although the interfacing is different, both processors attain the required high security level.**

***Keywords*-Hardware security; Crypto-processor; FPGA soft-core; NIOS II; MicroBlaze**

I. INTRODUCTION

Cryptographic hardware systems implement computationally extensive parallel cryptographic functions and complex sequential algorithms such as cipher modes, key management and cryptographic protocols. In order to fulfill contradictory speed/complexity requirements, sequential algorithms are often implemented using embedded general-purpose processors, while parallel functions are implemented in coprocessors. This approach is very frequent in asymmetric key cryptography [1], [2], and also in block ciphers (such as AES) hardware implementations [3], [4]. Some embedded systems using processor/coprocessor approach implement both symmetric and asymmetric key cryptography algorithms [5].

The use of processors weakens security in all the above mentioned solutions. In order to face side-channel attacks [6], [7], the keys must be changed/exchanged regularly using a key management protocol. When a general-purpose processor manipulates confidential keys, the keys are saved in the clear in processor registers or cache memory and are exposed to software attacks. One such attack has been recently demonstrated by Bangerter et al. in [8]. A small malicious software monitored the processor's cache memory during encryption and the confidential key was recovered remotely from the captured data within a few minutes.

In order to counter software attacks, the authors in [9] used two processors executing different tasks on different security levels. They created two virtual zones inside the physical memory: the protected memory zone was dedicated to private key storage and the unprotected one to public data memorization. The security processor had access to both zones and the general-purpose processor was allowed to access only the public virtual zone. Since both zones were located in the same physical memory, certain types of attacks, such as protocol attacks [10] or timing violation attacks, were still possible.

It is clear that software attacks targeting confidential keys can be countered only if enciphering is realized independently from the general-purpose processor (GPP), e.g. in a hardware cipher engine, and if the keys are stored in a dedicated memory. However, storing the keys in an external memory is not sufficient when facing software attacks. If confidential keys pass to the cipher via the processor in clear, they are vulnerable to attacks. In the proposed novel solution, the processor can manipulate the keys only indirectly: the keys are read/written from/to the key memory via a cipher and the processor can never read them in clear.

The paper is organized as follows: Section II proposes and discusses creation of hardware security zones that enable secure key management in conjunction with GPPs. Section III compares three basic ways of interfacing GPPs with external modules. Section IV describes a novel principle for the security extension of soft GPPs. In Section V, the proposed security extension is evaluated and validated on two implementations based on NIOS II and MicroBlaze processors. Results are presented in Section VI and they are discussed in Section VII. Section VIII concludes the paper.

978-0-7695-4529-5/11 $26.00 © 2011 IEEE
DOI 10.1109/FPL.2011.99

II. Principle of separation of protected and unprotected zones

We have explained in the previous section that to face software attacks on embedded systems using GPPs, processors shouldn't have access to confidential keys in the clear. It is thus necessary to isolate them from the key memory. In order to fulfill the highest security requirements, the separation should be realized on three levels:

- the protocol level,
- the architectural level,
- the physical level.

Next, we will describe these levels in more detail.

A. Separation on the protocol level

Integrity and confidentiality of exchanged keys is one of the most discussed security issues. In order to achieve a high security level, a robust communication protocol is needed. During key exchange, encrypted keys are usually transfered to the processor in packets. When a packet arrives to the processor, the keys are decrypted before being used for data encryption or decryption. However, if the decryption of the keys is carried out by the processor, these keys are exposed to software attacks. Therefore, keys have to be decrypted outside the processor in a dedicated unit so that the unencrypted key or it's fraction can never leave the unit. Moreover, keys have to be authenticated before being used. Once keys are decrypted and authenticated, they can be used for data encryption/decryption and authentication, but still outside the processor so that it will not have access to them. However, encrypted/decrypted data blocks can be processed by the processor e.g. when performing cipher mode operations.

It is the protocol that has to clearly separate key management and data processing tasks and define how and by which system units these tasks are performed. In addition to the separation of tasks, the protocol defines a multiple-level key hierarchy. Higher-level keys (i. e. master keys) are used to encipher lower-level keys (i.e. session keys). The highest-level key should be introduced to the system in unencrypted form through a separate entry by the trusted entity. All low-level keys should either be generated inside the security module using a True Random Number Generator (TRNG) or received in a packet and decrypted. The low-level keys are used for data encryption, decryption and authentication.

B. Separation on the architectural level

The principle of the separation on the architecture level is illustrated in Fig. 1. This principle is based on the creation of three zones: a processor zone, a cipher zone and a key zone. The processor exchanges data with a cipher through the data bus (in black in Fig. 1). Encrypted session keys are also transported through this data bus when being exchanged with other communication counterparts. No other way for accessing the key memory from the GPP must exist.

Unencrypted keys are stored in a dedicated memory situated in a key storage zone. The key memory is separated from the GPP by the cipher zone. The keys are transfered between the cipher block and the key memory via the key bus (in grey in Fig. 1). This bus must be completely separated from the data bus interconnecting the GPP with the cipher. It is essential that paths allowing unencrypted secret keys to pass from the key bus to the data bus must not exist. This precondition is one of the most important from the security point of view because it enables separation of the key storage and the processor zones.

Before enciphering/deciphering data blocks, the cipher is initialized with the selected key via the cipher key bus (in dashed grey in Fig. 1). Key selection is controlled by the processor through a control bus. The control path must be organized so that in case of an attack on the control bus, the selection of an incorrect key (e.g. by addressing a key out of the key address range) must be prevented.

The principle of the creation of security zones is independent from the type of encryption algorithm – any symmetric key block cipher, with or without side channel attack countermeasures can be used. Furthermore, the two separation walls delimiting the cipher can be used during dynamic reconfiguration of the cipher engine when changing cryptographic algorithms, to prevent changes to the key memory contents.

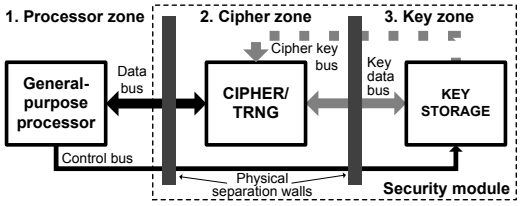

Figure 1: Separation on the architectural level

C. Separation on the physical level

To achieve a high quality physical separation of the processor and the key zones, separation has to be performed on two levels. First, the architecture of buses has to be organized in such a way that even after a physical attack, keys from the key zone cannot pass unprotected into the processor zone. Bus multiplexers directing the flow of data to/from the cipher have to be placed so that even if their control is violated, no physical path for keys to escape from the key zone can be created.

In order to achieve a higher level of physical separation, an isolated area of the chip has to be dedicated to each zone. On the border of any two neighboring zones, an empty area has to be used. This area represents an insulation wall between neighboring zones (see Fig. 1). Only selected signals are allowed to cross this wall. This countermeasure minimizes the possibility of the loss or corruption of secret keys by residual electromagnetic radiation from the

protected zone (i.e. key zone) to the unprotected zone (i.e. processor zone). Using this approach, a physical attack on one zone has a minimal impact on the other zones. This physical insulation principle is recommended by NSTISS in its Red/Black installation guidance [11].

III. Communication interface

The data interface between the security module and the processor plays a very important role in the system design. When considering the architecture, a trade-off has to be found between the performance, area and security criteria. Unfortunately, this can often lead to contradictory requirements.

A. Interface properties

Overall system performance depends on the interface type and parameters, such as bus width and latency. When managing keys, small data blocks are exchanged between the processor and the security module and pipelining is not efficient. In order to achieve higher performance, it is the best to suit the bus width to the cipher width. Unfortunately, this is not always possible.

From the security point of view, point-to-point communication is in general more secure than point-to-multipoint commnunication, because data are exchanged only between two units, and other peripherals are not physically connected. When using a point-to-multipoint interface, the bus is shared among all communication counterparts, and data exchanged between the security module and the processor can be potentially eavesdropped by other peripherals. In this case, some techniques, such as small firewalls protecting each peripheral on the bus [12], can be used.

B. Interface types

The security module can be interfaced with a GPP using the following types of connections:

1) The internal processor bus: The security module is included in the processor data path. Data passes to the module directly from registers as operands. Results are returned to the registers. The security module is controlled directly by the processor control unit through a dedicated control bus. The advantage of this solution is its high performance and minimal latency. Unfortunately, the module is a part of the processor's critical path and therefore it can slow the whole system down if not properly designed. A high security level is naturally achieved by the point-to-point connection.

2) The coprocessor-like bus: The security module is not included in the processor data path, but it is connected through a fast internal bus (often a coprocessor bus), running mostly at the same clock frequency as the processor core. This bus enables direct access to the processor registers thus minimizing communication latency, although the latency is higher when compared to the previous alternative. The connection has a point-to-point nature therefore a high security level is maintained.

3) The peripheral bus: The security module is connected to the processor through a bus using a point-to-multipoint connection. In the best case, the module is connected directly to a high-performance system bus, otherwise data are transferred across one or several bus bridges increasing the latency. Furthermore, the security module cannot be controlled by a dedicated control bus and instructions must be fetched across the same bus as the data. Because of these disadvantages, this type of connection will not be considered in this paper.

IV. Implementation of the security module

The implementation of the security module is illustrated in Fig. 2. Three zones (the processor, the cipher and the key storage zone) can be clearly distinguished. Three type of buses are used: he data bus (in black), the key transfer bus (in grey) and the cipher key bus (in dashed grey). It is important that key buses never pass through the processor zone and data buses never pass through the key storage zone. The module is organized so that secret keys can never leave the key zone without passing through the cipher. This strict separation on the architectural and physical level guarantees a very high security level.

As presented in Sec. II, any encryption algorithm can be used in the security module. In order to validate the principle, we use a 128-bit Advanced Encryption Standard (AES) core. Input data, keys and output data are registered in cipher block input/output registers. The cipher engine has a 128-bit folded data path, and encryption is completed in 11 clock cycles.

Keys are organized in two hierarchical levels. High-level master keys for encryption and authentication are stored in the master key register. These keys are initialized via dedicated key input during system initialization by the trusted entity. Master keys are used solely for encryption, decryption and authentication of low-level session keys. Session keys are generated inside the module by a TRNG (any principle can be used) or received from the processor and decrypted and authenticated using master keys. When generated inside the module, keys are post-processed in the decipher core. Session keys are used only for encryption/decryption (using cipher modes) and authentication of data (e. g. using CBC-MAC mode).

The authentication of keys is supported directly inside the module by including a comparator (CMP) that is responsible for comparison of digital fingerprints. The security module is controlled by a local control unit (CTRL) that interprets the fetched instructions. Comparison results or cipher status flags are saved in the status register, which can be read by the processor. This principle of demand-response dialog permits the secure and high-performance operation of the module.

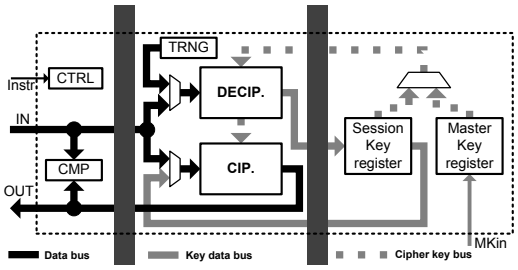

Figure 2: The security module implementation

A. Communication algorithm

Before establishing the communication channel between two parties A and B, master key registers of the communicating security modules have to be initialized with the same encryption and authentication keys by a trusted entity.

Subsequently, the device that starts the communication has to generate a session key. The session key is generated by the TRNG inside the security module on the communication side A. The key is then post-processed in the decipher, using a master key, and stored in the session key register. Afterwards, a digital fingerprint is generated by enciphering the generated key using an authentication master key.

When both the session key and its fingerprint are generated, the data blocks can be encrypted using the session key and sent to the GPP. The GPP implements cipher modes by processing subsequent data blocks according to the selected algorithm. Finally, the processor creates a packet. This packet contains a generated session keyK_s, its fingerprint P_s, encrypted data and a Message Authentication Code (MAC) of the packet. At the end of the transaction, the packet is sent to communication partner B.

Processor B receives the packet, recognizes the packet header and extracts the encrypted session key with its digital fingerprint P_s. The key is sent to the security module, where it is decrypted using a master key and stored in the session key register. The fingerprint R_s of the session key is computed by enciphering it with a master authentication key. The processor sends the received fingerprint P_s to the security module, where it is compared with the computed fingerprint R_s. If the fingerprints match, the receiving processor can decrypt data using the acquired session key by executing selected cipher mode operations.

V. Extension of selected soft-core processors

Proposed security extensions were implemented in two FPGA families using Altera NIOS II and Xilinx MicroBlaze soft processors. Each processor system included the same security module that was connected to the processor using a wrapper, which was compatible with the processor interface. The wrapper is responsible for the translation of the control commands and the bus width conversion (a 128-bit security module data bus is transformed into a 32-bit processor bus). Both communication interfaces use point-to-point communication increasing device security.

A. NIOS II with the security module extension

Implementation of the NIOS II processor with its security extension is illustrated in Fig. 3. All security module operations are implemented as custom instructions. Data are transfered from the processor register in 32-bit words via the wrapper. When the instruction execution is finished, data from the security module is sent back to the processor, again in 32-bit blocks.

Because of the use of custom instructions, the NIOS II control unit drives signals that control directly the operation of the security module. This direct connection eliminates an unwanted latency increase, thus accelerating the execution of custom instructions.

Since the security module is included in the NIOS II data path, the critical path of the processor is extended by the data path of the security module. This directly affects the processor's maximum clock frequency.

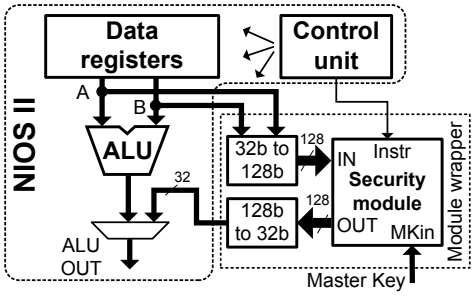

Figure 3: Interfacing NIOS II with the security module

B. MicroBlaze with the security module extension

Implementation of the MicroBlaze processor with its security extension is illustrated Fig. 4. In contrast to the NIOS II processor, a custom instruction set implementation is not possible. However, the MicroBlaze architecture supports a high-performance Fast Simplex Link (FSL), which allows interfacing external modules with processor registers. The FSL bus is 32 bits wide. So 128-bit data blocks have to be divided into four 32-bit blocks before being transfered to the security module.

Unfortunately, control signals cannot pass directly from the MicroBlaze control unit to the security module. Instructions are sent to the module via the FSL data bus and FIFOs. Before each operation, one 32-bit instruction word has to be sent. However, this operation imposes an additional instruction on the program code, which slows down the code execution and data exchange between the processor and the security module. On the other hand, the security module is not a part of the processor's critical path, thus the clock frequency of the processor is not affected. Despite the fact that FIFOs insert additional latency, they enable

separation of processor and security module clock domains, thus security module can run on a higher clock frequency than the complex processor.

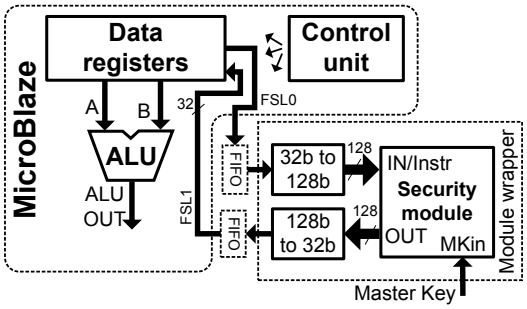

Figure 4: Interfacing MicroBlaze with the security module

VI. Results

The two processors with their security extension modules presented in the previous section were described in VHDL and mapped to two FPGA families. The NIOS II system was implemented in the Altera NIOS II evaluation board featuring a Stratix II series device, the EP2S60F672C5ES. The project was compiled and mapped to the selected device using Quartus II version 9.2. The MicroBlaze system and its extension were implemented in the Xilinx ML605 evaluation kit featuring a Virtex 6 series device, the XC6VLX240TFF1156. For synthesis and mapping, ISE version 12.4 was used. A small hardware module including a Cypress EZ USB interface was connected to both evaluation boards for data transfers from/to the PC.

The implementation results concerning the logic area and memory requirements are presented in Table I. The area is expressed as the number of occupied Adaptive Logic Modules (ALM) for the Altera family and as Slices for the Xilinx family. For comparison, we recall that one ALM in the Altera Stratix II family contains two 4-input Look-up tables (LUTs) and two flip-flops (FFs). One Slice in the Xilinx Virtex 6 family contains four 6-input LUTs and eight FFs. Therefore, the results cannot be directly compared. The memory requirements are given in kbits for both technologies. Besides the processor and its security extension, a small module containing a data interface to the external Cypress USB interface device was embedded in both systems. It was used only for testing purposes and it does not constitute an inherent part of the system. It is therefore not included in Tab. I. For clarity, we present the results for the processor and for its extension separately. The results are discussed in the next section.

In order to compare the achieved throughput fairly, the clock frequency of both systems was set to 50 MHz. The throughput was evaluated by transferring packets from the PC to the FPGA (and vice versa) via the USB interface. Each packet contained an encrypted session key, its digital fingerprint and five 128-bit payload blocks. Packets were analyzed in the processor, which then sent the session key and its fingerprint to the security module. Once the key was decrypted and authenticated, the processor sent data blocks to be decrypted. Subsequently, the processor recreated new packets containing the received data and sent them back to the PC. When implementing complete protocol, the NIOS II-based system achieved the overall throughput of 25,1 Mb/s and the MicroBlaze-based system achieved 18,4 Mb/s.

Table I: FPGA resource utilization

	Extended NIOS II		Ext. MicoBlaze	
	ALMs	**RAM kb**	**Slices**	**RAM kb**
System total	2531	243.9	1954	1206.0
→ Processor	1204	187.9	1350	774.0
→ Sec. module	1327	56.0	604	432.0
Extension overhead	110.2%	29.8%	44.7%	55.8%

The security of both solutions depends on the type of interface between the security module and the processor. Since the separation of the key and processor zones was achieved on the protocol and architectural level, both implementations should be robust against software and timing violation attacks. In order to validate this assumption, preliminary software attacks were implemented in tests by experimental reordering and/or replacing of instructions. To introduce faults into the control logic, a preliminary timing violation attack was carried out by increasing the clock frequency over its maximum allowed value. These tests confirmed the robustness of the proposed techniques of separation that were shown to be secure-by-design. We consider this evaluation as a preliminary security estimation and further extensive tests should be carried out in the future.

VII. Discussion

Area requirements for NIOS II and MicroBlaze extensions from Tab. I seem to be different. This is due to differences between ALMs and slices in Altera and Xilinx families and also because of the size of the soft processors. In the Altera FPGA, the security module area is similar to that of the NIOS II processor (1327 versus 1204 ALMs giving 110%). However, since the security module included both AES cipher and decipher cores, we can conclude that the security extension cost due to zones separation is negligible. On the other hand, the MicroBlaze processor occupies bigger area and the security module overhead is only 30%. The separation cost remains negligible.

As presented in Section VI, the MicroBlaze processor with its extension achieves 73% of the throughput of the NIOS II implementation. This is caused by the more complicated communication protocol (data and control words) across the FSL bus in MicroBlaze, compared to the straightforward custom instruction implementation in NIOS. This difference could be reduced if data were transferred to the

security module using DMA transfers. This way, the FSL bus would serve only to transport instructions to the security module.

As required, preliminary attacks that were carried out against the processor implementations were not successful. The implemented protocol and architectural separations of the key and processor zones were therefore shown to be effective. Furthermore, a higher security level could be achieved if the security zones were physically isolated on the chip. Although this physical isolation by insulation walls was not realized, it can be easily done in the future because of existing strict separation of logical modules.

The principle presented in this paper concerns two soft GPPs. We believe that it can be extended to other GPPs, thus increasing significantly their security. This approach is very practical, because of the short design time and low costs. However, faster and perhaps more secure implementations could be obtained by creating a specific-purpose processor such as the one published in [13] taking advantage of a reduced instruction set dedicated to cryptographic operations. Nevertheless, this custom processor would have to comply with all the above mentioned separation techniques and the principle is thus generalizable.

Since encryption standards are always evolving, it is important that the cipher implementation inside the security module be made easy to update. One of possible solutions could be the use of a dynamic reconfiguration technique. This technique allows for creation of a special dynamically reconfigurable zone reserved for the cipher core. This option helps to avoid expensive hardware upgrades. Moreover, the physical separation techniques are similar to those used in dynamic reconfiguration procedure. A zone that is dynamically reconfigurable can be very easily isolated from surrounding modules by an empty space created around it serving as an insulation wall.

VIII. Conclusion

We have proposed a novel principle allowing general-purpose processors to operate with secret keys in a highly secure way. The principle is based on creation of separated processor, cipher and key zones. Separation is implemented on the protocol, architectural and physical level, and it guarantees that unencrypted keys can never be transfered from the key zone to the processor zone. The only way to transfer the keys to the processor zone is through the cipher zone: they are encrypted before entering the processor zone and must be decrypted when entering the memory zone. The proposed solution enhances security substantially when compared with existing soft-core cryptographic extensions.

The separation principle was implemented in FPGAs and tested using both NIOS II and MicroBlaze processors. The obtained throughput, including the processing of packets, key management and data encryption/decryption and authentication was about 25 and 18 Mb/s, respectively. This speed was limited mainly by processors and their data interfaces. The area of the system increased by 110% when compared with the smaller NIOS II processor and by 44% when the MicroBlaze processor was taken as a basis.

Acknowledgment

The work presented in this paper was realized in the frame of the SecReSoC project number ANR-09-SEGI-013, supported by the French National Research Agency (ANR).

References

[1] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF (2^n)," *IEEE Transactions on Computers*, pp. 1269–1282, 2007.

[2] M. Machhout, Z. Guitouni, K. Torki, L. Khriji, and R. Tourki, "Coupled FPGA/ASIC Implementation of Elliptic Curve Crypto-Processor," *International Journal*, vol. 2.

[3] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single-chip FPGA implementation of a cryptographic co-processor," in *2004 IEEE International Conference on Field-Programmable Technology, 2004. Proceedings*, 2004, pp. 279–285.

[4] Y. Eslami, A. Sheikholeslami, P. Gulak, S. Masui, and K. Mukaida, "An area-efficient universal cryptography processor for smart cards."

[5] M. Hani, H. Wen, and A. Paniandi, "Design and implementation of a private and public key crypto processor for next-generation it security applications," *Malaysia Journal of Computer Science*, vol. 19, no. 1, pp. 29–45, 2006.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *CRYPTO99*, pp. 789–789, 1999.

[7] F. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J. Quisquater, "Power analysis of FPGAs: How practical is the attack?" *FPL'03*, pp. 701–711, 2003.

[8] E. Bangerter, D. Gullasch, and S. Krenn, "Cache games–Bringing access-based cache attacks on AES to practice," *Workshop COSADE*, pp. 215–221, 2011.

[9] A. Ashkenazi and D. Akselrod, "Platform independent overall security architecture in multi-processor system-on-chip integrated circuits for use in mobile phones and handheld devices," *Computers & Electrical Engineering*, vol. 33, no. 5-6, pp. 407–424, 2007.

[10] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors - a survey," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006.

[11] J. M. McConnell, "TEMPEST/2-95," *NSTISSAM*, 1995.

[12] P. Cotret, J. Crenne, G. Goniat, J.-P. Diguet, L. Gaspar, and G. Duc, "Distributed security for communications and memories in a multiprocessor architecture," *Workshop RAW*, 2011.

[13] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "HCrypt: A Novel Concept of Crypto-processor with Secured Key Management," *ReConFig'10*, pp. 280–285, 2010.